Labelling Game for Twitter Streams

Semester Project Report

Hanser Valérian

Fall 2010

The main goal of this project is to build a mean and to populate a database with correct information regarding some decided pairings. This database will then be compared to the results of algorithms that process the same pairings, which provides a way to assess algorithms' performance.

The main focus is set on the following pairing, tweet messages and company. For a given set of tweet messages, we should be able to decide if either the word refers to the given company or not. In order to populate the database with such information, a web game application is used, which will allows any kind of users who wishes to play to give its feedback. We record those feedbacks and when enough feedbacks are collected, we may be able to build a subset of the correct information we want based on the feedbacks.

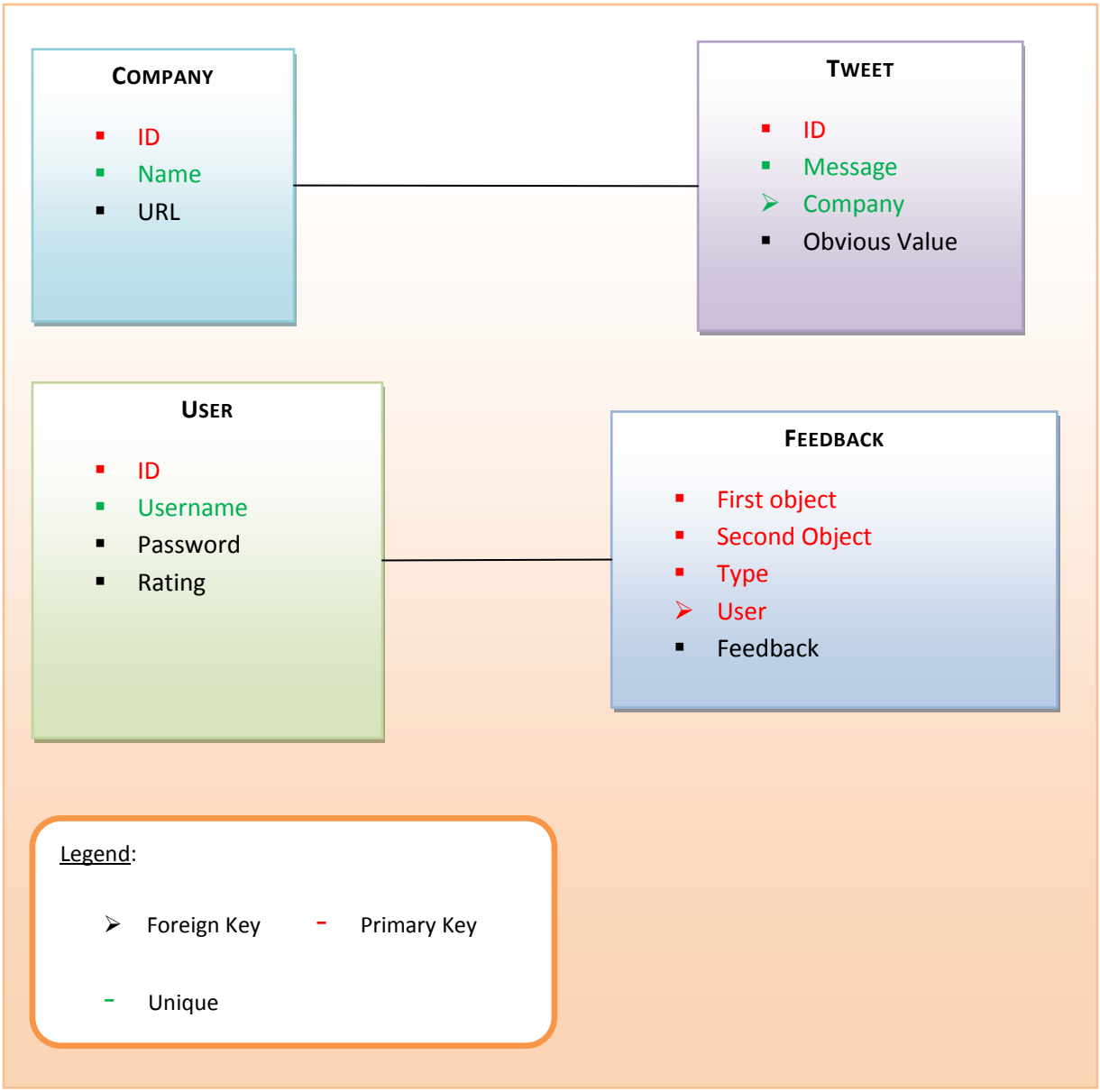
This work is about developing such an application, and, to do so, it was suggested to use a web framework named Django:

"Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design."

As said, Django uses the well-known programming language Python and has many built-in tools for ease of use. For example, one can define its whole database in python, and Django provides an API for accessing this database, which can be MySQL, SQLite3, Oracle, etc...

First of all, we will discuss in-depth about the database used in this project, how it is built, how it is implemented and the choices made concerning the design. In a second time, we will see how the client side web application works, as for the database, we will discuss about how it is done, the problems encountered and the choices made regarding the design. Finally, we will take a look at the administrator functionalities.

Database



Game Database Schema

The above schema briefly describes the final database used for the game itself. The database access in this project is handled by the Django API; all the tables are defined as 'Models' in Python code. The database engine used is MySQL because of the knowledge already acquired about this engine.

We will now give a complete definition of this database, table by table, explaining what the different fields are, the constraints of using Python - that is what could have been done better in RAW MySQL - and the relevant alternatives designs that were thought about.

User

Responsible for representing a player account, this table has four fields:

- The first one is the primary key **'ID'** that is an auto-incremental integer.
- The second one is the **'Username'** that is a field of characters, which is used as the login for a player; this field is tagged as unique; the maximum length is set to 16.
- The third one is the **'Password'** that is a field of characters that contains an encryption of the user's password; the maximum length is set to 32.
- The last one is the **'Rating'** that is float, corresponding to a grade that is defined on the player feedbacks; default rating is set to 0.0.

As the player can choose its *'Username'*, it has to be unique; we can't ask a player to remember an auto-attributed ID and we want to be able to differentiate two different players. We could have used the field 'Username' to be the primary key, but integers are lighter to handle.

The 'Rating' is based on the user feedbacks, we can determine a lot a policies to compute the rating, as that is not a primary concern in this project, we used a simple way to try determining if the user answers randomly to the suggested tweet-company pairs.

Tweet

Responsible for representing a tweet message, this table has four fields:

- The first one is the primary key **'ID'** that is an auto-incremental integer.
- The second is the **'Message'** that is a field of characters, which contains the tweet message we want to evaluate.
- The third one is the foreign key **'Company'** to which we want to know if the message refers to or not, the foreign key refers to the *Company.ID* integer.
- The last one is the **'Obvious Value'** that is a field of character, it is to know if the message surely refers to the company or not or unknown; the default value is 'Unknown'; the maximum length is set to 1, *see below for details*.

The pair *'Message'-'Company'* is unique together because we want to be able to have a same text message to belong to different companies and avoid duplicated, even if this not likely to happen.

The *'Obvious Value'* should in fact be an ENUM value in MySQL, but the python defined models doesn't allow it directly. Instead, it is a field of character of maximum size 1 to which is associated an array of choices.

Company

Responsible for representing a Company, this table has three fields:

- The first one is the primary key **'ID'** that is an auto-incremental integer.
- The second is the **'Name'** that is a field of characters, which contains the name of the company; this field is tagged as unique; the maximum length is set to 30.
- The last one is the **'URL'** that is also a field of characters, containing the URL of the company; the maximum length is set to 200.

The field 'Name' is unique because we don't want two different companies to be named the same way for consistency concerns. We could have used the field 'Name' to be the primary key, but integers are lighter to handle and due to design concerns regarding the feedback table that will be discussed later.

Feedback

Responsible for representing a feedback for any pairs, this table has five fields:

- The first one is **'First Object'** that is an integer, which refers to a table.
- The second one is **'Second Object'** that is an integer, which refers to a table.
- The third one is **'Type'** that is a field of characters, to know what kind of pair the row belongs to; maximum length is set to 1, *see below for details*.
- The fourth one is the foreign key **'User'** that refers to the *User.ID* integer the feedback was given by.
- The last one is **'Feedback'** that is a field of characters, which is the feedback of the user for the pair-type; maximum length is set to 1 *see below for details*.

The feedback table is able to handle any kind of pairs as long as each side the pair is differentiable among itself by integers, a primary key for example. Each raw represents a user's feedback for a pair of a certain type. The tuple *('First Object', 'Second Object', 'Type', 'User')* is unique together and hence form the primary key in Python, in the MySQL database the table has an auto-incremental integer field ID for primary key.

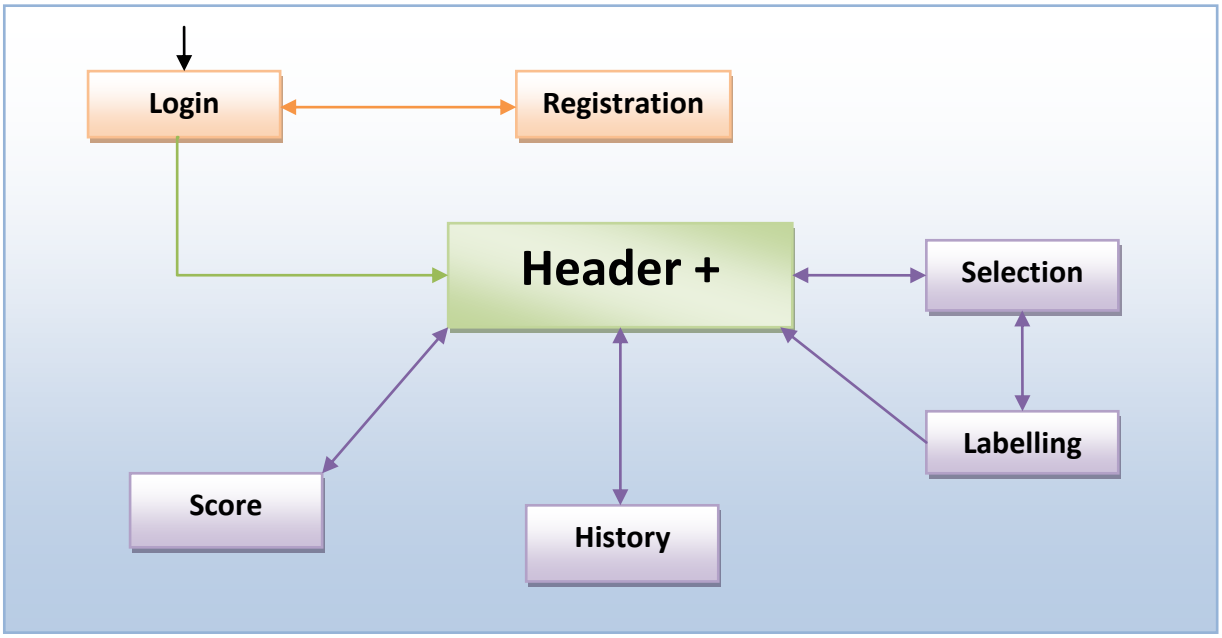
The *'Type'* and *'Feedback'* should in fact be an ENUM values in MySQL, but the python defined models doesn't allow it directly. Instead, it is a field of character of maximum size 1 to which is associated an array of choices.

In the case of the *'Tweet'-'Company'* pair, we could have used only the tweets' id and retrieve the company via its foreign key, needing only one field in this table; but we wanted the Feedback table to be able to represent any pairs, it was not build for the tweet game specific needs.

This table has to be used with caution, as there is no foreign key regarding *'First Object'* and *'Second Object'* because in one hand we don't want this table to be limit to one kind of pair,

in the other hand we don't have foreign key fields that are *'Type'* dependant as this is not provided by the engine. So, this must be handled on a higher level, considering the *'Type'* and populate the database accordingly.

Web Application - Client Side



Web Application Chart

The above schema is the link chart for the client side of the web application. The website is Django powered, that is, when we enter an URL, it will be matched to some patterns through regular expressions, if there is a match; it will call a Python function that is in charge of dealing with the client request. Most of those functions in this project will return directly a webpage to be rendered. All the pages are using HTML and Django own defined syntax that can be seen as an alternative, complement to PHP.

We will now give for each page what they contain, what functions are called and their purpose, the problems encountered and the choices made.

Login

This page is the first page displayed when we enter the server URL/IP. The page is set to display an error message if there is such message.
It also contains an HTML POST form in which the inputs are Django designed; there is a python definition for this form, it defines a username and a password; maximum length is set to 16 and 20 respectively. We use this Django powered form because of the automatic form validation it provides. So this form is used to authentificate yourself with the server. There is also a link to the registration page for those who don't have an account yet.

The function in charge of this page is divided in two parts. If you access the URL/IP with a "GET" request – which is the default when you enter the URL/IP -, it will simply display the login form and the link described above. When the POST button of the page is clicked, we go to the same URL and hence to the same function, but this time the request method is set to

"POST". This time the function will try to validate the POST request, if successful, we retrieve the account in the database (table User) with the given username and we compare the encryptions of the account's password and the password given in the form. We display the login page with an error message if the password is not the same or if there is no account for the given username. If all is successful we retrieve all the companies – *see Menu for more information* - and display the menu page.

One of the problems here is to secure the user's password, we wanted the password to be encrypted client side and then sent through the network, but we didn't find an easy way to do so, and as security is not a primary matter in this project and the consequences are minor, it simply send the password in clear and we encrypt it when received by the server.

Registration

This page is displayed when we click on the 'Register' link on the login page. The page is set to display an error message if there is such message.
It contains the same HTML POST form as the login page as the two have nearly the same purpose.

In the same way as the login function, the register function display the form in case of a "GET" request. When the request method is set to "POST", the function will try to validate the POST request; if successful we try to retrieve the account in the database with the given username. If successful we display the registration page again explaining the user that this username is already taken (via an error message), else we simply create a new account with the provided username and encrypted password and insert it into the User table and display the login page.

Selection

This page is accessed after the user has entered the username and password. The page is set to display an error message if there is such message.
This menu is composed of a complex HTML POST form, it is actually a company selection, it displays all the companies in the database for the user to choose to play with. The displaying of those companies is done by a Django specific syntax loop over the companies received as a page argument.

Regarding the function called by the company selection, it tries to retrieve the select company name, if successful it directly call another function that is described is the Labelling section. If no company is selected it display the selection again with an error message.

Header

The header is available once logged in and is available on every pages; it allows accessing directly to the 'Score', 'History' and 'Selection' pages. The three buttons are in fact POSTs

which only input is the hidden input which value is set to the actual user's username, this way we can know which user requires to see its score, history or company selection, *see the corresponding pages for more details*.

Score

This page is display the score of the player for the overall companies.

So as to computing the score, we retrieve the account ID (User.ID field) for the given username which was received via the POST request hidden value discussed in the Menu section. Once we have this User.ID, we can retrieve all the feedbacks given by this user for the pairing *'Tweet'-'Company'* and compute a score based on those feedbacks. Here again, there are many policies to compute a score, but this is not of primary concern in this project, so the score is computed by adding how many players have answered the same way as the user for each tweet. If there is no feedback given by the user, a message explaining this is displayed.

History

This page displays all the feedbacks the user has given from the creation of the account. The page contains a HTML table coupled with a Django powered loop over a table received as a page argument.

The function purpose is to build the table that will be displayed. The table is an array of associative table, we retrieve the account ID (User.ID field) for the given username which was received via the POST request hidden value discussed in the Menu section. Once we have this User.ID, we can retrieve all the feedbacks given by this user for the pairing *'Tweet'-'Company'* and loop over this set, retrieving for each the *'Company.Name'*, the *'Tweet'* and the *'Feedback'* putting them in an associative table and appending it to the main table. If there is no feedback given by the user, a message explaining this is displayed.

Labelling

This page is accessed after selecting a company in the Menu. The page is set to display an error message if there is such message.

Using a form and Django powered loops over the set of five tweets given as a page argument, it displays the five tweets which associated company (*'Tweet.Company'*) is the one selected. For each five tweet, it displays the tweet message and three choices: 'Yes', 'No' and 'Unknown' for the user to give its feedback, only one choice can be select for each tweet. When the submit is done, a validation function is called and a new set of tweets is displayed.

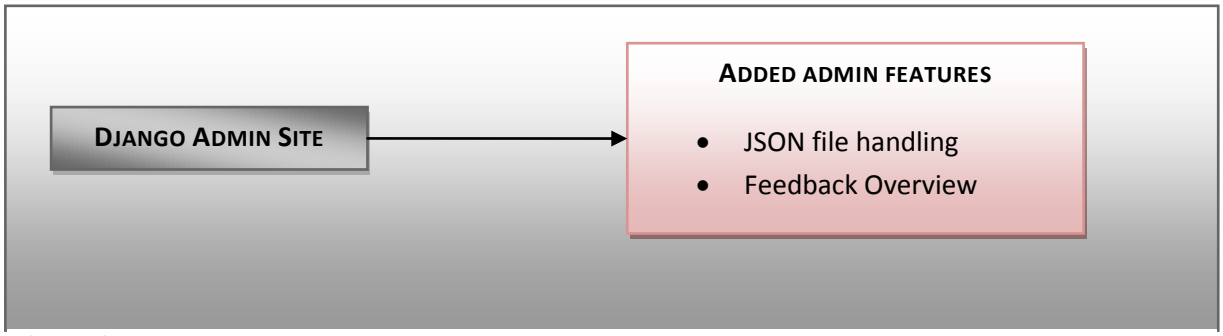
In this page, two big functions are used. The first one is the one that build the set of five tweets to be displayed.
There are many policies that could have been used in order to select five tweets among the existing tweets for a given company. The main concern regarding this problem in this project is to have a minimum of feedbacks for a tweet in order to decide either or not it refers to the company. So in one hand, we should select tweets for which a feedback has already happened. In the other hand, we also want most of the tweet set to be played with. The policy we decided to apply here is to select the tweets ordered by ID (which is done by default by MySQL) and take the five first tweets that don't have more than five feedbacks. This is an advanced query that the Django API could not handle directly. Fortunately, it allows us to make raw SQL queries; this feature has been used to perform such a query. If the result set is empty, we simply inform the user that there are no more tweets to be labelled for this company.

The second big function is the processing of the user feedbacks when he wishes to submit its feedback.
The value in the POST contains the tweet ID and the user feedback. Hence, we must retrieve that two data. In order to do this, we use regular expression's groups, we use the regular expression "^(d+)([U|Y|N])$". First we store all the values in a table, and then loop over this table and we create the set of feedback corresponding to the current user, the company he selected, the tweet ID and user's feedback obtained with the regular expression. Once this is done we continue with the next five tweets, calling the above described function.

Security Matters

Apart from the password clear transmission, one may worry a little about security, as seen, the entire site is done with the POST method. Hence the first matter is to avoid Cross Site Request Forgery. Fortunately, Django provides a protection against this attack using a cookie set to a random value.
For more information, see Cross Site Request Forgery protection.

Web Application – Admin Side



Admin Side Overview

The application management is mainly handled by Django built-in Admin Site. You have to login on the admin website in order to access the provided tools, the username and password are different from the ones used for the game. The Django admin site was design to support a multiple websites so it has a lot of tables regarding permissions like super-user, website specific admin, etc...

Its flexibility allows us to customize and add some features to the basic site, we can set up filters for the database, adjust the way it is displayed. Here we will only discuss about the two added features: the Feedback overview and the JSON file handling.

Feedback Overview

The feedback overview is a page which contains a table that display all the tweets message using Django powered loop over a page argument for a given company and a summary of how many players answered 'Yes', 'No' and 'Unknown' and finally an appreciation of the overall feedbacks for each tweet.

The function responsible for building such a table is done as follow: the main of the function's work is handled by a raw SQL query using a "GROUP BY message, feedback" and the COUNT clause. The result is that it creates a row for each *'Message'-'Feedback'* pair and count how many such pair exists in the *'Feedback'* table. Hence we only have to handle the pairs where count is zero has it will not be an entry in the result set. The table the function built is an array of associative table, so we store in an associative table the information of the message, the feedback and its count, which will be appended to the main table, given as argument to the webpage.

The policy regarding the appreciation for the tweet message is the following:

- #'Yes' > #'No' => 'Yes'
- #'Yes' < #'No' => 'No'
- #'Yes' = #'No' => 'Unknown'
- #'Unknown' > #'Yes' and #'No' => 'Unknown'

JSON File Handling

The page contains a POST form, giving in a '.json' file path.

*"**JSON** (an acronym for **JavaScript Object Notation** pronounced /ˈdʒeɪsən/) is a lightweight text-based open standard designed for human-readable data interchange. It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for virtually every programming language." Source: http://en.wikipedia.org/wiki/JSON*

Such a file can be directly received from the Twitter API according to some keywords.

The function receiving this file path will try to open the file. If the file exist, it takes the first characters of the filename as the company name (suppose the filename is Company.json or Company-234.json, the company name will be Company is both cases) using the regular expression "'^.*/([a-zA-Z]+)-?[0-9]*\.json$", and then create the company if it doesn't exist. We make some text processing over the file, like taking care of blanks, line returns and quote flags. Unfortunately, due to encoding reasons, some strange characters may be badly converted to UTF-8, resulting in some non-fluent messages (i.e. appearance of strings like "u286"). According to the associative side of JSON files, we can access to the tweet messages as we access a table, so we can populate the Tweet table with the corresponding company. As the *'Tweet.Message'* and *'Tweet.Company'* are unique together, we will not insert duplicate messages. Aside from populating the database, we build an HTML text with all the messages or a duplicate message when needed that will be displayed to the admin for him to know that the work is done and inform him if duplicates were found.

What I have learned

During this project, I first have re-used and somehow deepened my knowledge about databases, especially MySQL. In second term, I have learned to use a nice web framework, how to use the tools it provides, and modifying some part of it. That framework uses Python, which I never practiced before, this language was new to me and as far as the project needed it was quite ok to learn. In the beginning, I got confused between Django own syntax and Python. One of the longest works was reading Django documentation regarding models (that how the name database tables) and trying to match those models to good MySQL definitions. Also, I hardly ever wrote web pages, HTML was another language to deal with, and I think I only got the basics of it. Many other website aspects were attractive, as for building nice looking pages, securing data transmissions, but this requires other fields to know as CSS, Jscript etc... And this could be the purpose of an entire project.